4. Implement a facility which accepts descriptions of problem solver class and enables the user to ask the questions for that class about an example system;

5. Investigate new kinds of explanation capabilities -- for example, how a program's operation might be meaningfully summarized for several kinds of users, such as domain experts and programmer/system designers.

References for this section

[Buchanan71] Buchanan B G, Lederberg J, The heuristic DENDRAL program for explaining empirical data, IFIP, 1971, pp 179 - 188.

[Buchanan72] Buchanan B G, et.al., Heuristic theory formation: data interpretation and rule formation, in Machine Intelligence 7, (Meltzer & Michie, eds), pp 267-292, 1972.

[Davis76] Davis R, Applications of meta level knowledge to the construction, maintenance, and use of large knowledge bases, (thesis), AI Memo 283, Stanford University, July 1976.

[Davis77] Davis R, Buchanan B, Shortliffe E, Production rules as a representation for a knowledge-based consultation program, Artificial Intelligence (to appear, Jan 77).

[Engelmore77] Engelmore, R, and Nii, H Penny, A Knowledge-Based System for the Interpretation of Protein X-Ray Crystallographic Data, 1977.

[Lenet76] Lenat, D, AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search, Ph.D. Thesis in Computer Science, 1976.

[Lesser74] Lesser V R, Fennell R D, Erman L D, Reddy D R, Organization of the HEARSAY II speech understanding system, IEEE Transactions on Acoustics, Speech, and Signal Processing, ASSP-23, February 1975, pp 11-23.

[MACSYMA74] The MACSYMA reference manual, September 1974, The MATHLAB Group, MIT.

[Nii77] Nii, H. Penny and Feigenbaum, E, Rule-based Understanding of Signals, presented at Workshop on Pattern-directed Inference Systems, 1977.

[Newell72] Newell A, Simon H, Human Problem Solving, Prentice-Hall, 1972.

[Nilsson71] Nilsson N J, Problem Solving Methods in Artificial Intelligence, McGraw Hill, 1971.

[Pople75] Pople H, Meyers J, Miller R, DIALOG, a model of diagnostic logic for internal medicine, 4IJCAI, pp 848-855. (the system has recently been renamed INTERNIST).

[Shortliffe76] Shortliffe E H, Computer-based clinical therapeutics: MYCIN, American Elsevier, 1976.

[Stefik77] Stefik, M and Martin, N, A Review of Knowledge-Based Systems as a Basis for a Genetics Experiment Designing System, 1977.

3.3.2 SOFTWARE EXPORT ALTERNATIVES

Over the past few years, a number of the programs being developed by SUMEX-AIM projects have reached a developmental maturity where we need to consider ways of meeting the demands to make them operationally available to a larger user community and to export them where appropriate to other sites. Current examples of such programs include the CONGEN biochemical structure elucidation program, the SECS chemical synthesis analysis program, and the MYCIN, ONET, and INTERNIST medical diagnosis programs. Our present PDP-10 facilities are quite insufficient for meeting the operational needs of this growing group of users, even if providing this level of service were within the SUMEX-AIM mandate.

These programs have been written in a variety of source languages (principally various dialects of LISP or SAIL) and are characterized by very large address space requirements. The development medium for these programs at Stanford has been the PDP-10 TENEX environment and the choice of language made to facilitate development and representation of logical program concepts. In contemplating the export of such programs, several points seem relevant:

- Development is continuing on the programs to extend their conceptual framework and operational effectiveness. This implies that there must be a low threshold between developmental versions of the programs and operational ones during this phase and that the implementation environment of the programs must be conducive to both.

- Because of the complexity of the programs, it is likely that their maintenance and upgrade should be centralized. This implies a convenient means of receiving user feedback and of providing program updates.

- Because of the address space requirements for these programs (even after possible rewrite for increased efficiency), it does not appear reasonable to export them via 16-bit mini computers where unwieldy overlay structures would be required to circumvent the addressing constraints.

- The target community for these types of programs will be fairly heterogeneous. Users may include academic research groups, industrial houses, hospitals, and educational institutions. One can expect the native computing resources in these various user sites to cover a wide range of hardware and operating systems, not all existing PDP-10's. We cannot expect many users interested in the programs to be able to set up a full-scale PDP-10 site capable of running them.

We have been considering a number of mechanisms for exporting such software. These include a) implementing individual programs on machines which could be accessed by interested users over some (commercial) network, b) implementing or (reimplementing existing) individual programs in an appropriate language which is "machine independent" and thereby could be run on a user's existing computer given some minimum size, or c) making the programs available on an exportable machine (PDP-10 or its more cost-effective descendants) which is compatible with the existing programs and the centralized PDP-10 facilities used for continuing development.

3.3.2.1 NETWORK ACCESS

There is a growing number of uses of computer networks for program dissemination ranging from business accounting and modeling packages available from commercial vendors to attempts to consolidate research tools such as a collection of mass spectral library search and analysis programs (see for example S. R. Heller, G. W. A. Milne, and R. J. Feldmann, "A Computer-based Chemical Information System", Science, Vol. 195, Number 4275, page 253, 1/21/77). The existing network connections at SUMEX are well-configured for experiments within our capacity on this means of disseminating software. For many such programs, this seems to be well-suited for export; and indeed Heller reports 162 current user groups subscribing to his Chemical Information System. However, unless the network machine runs the same operating system and language in which the program was developed, a conversion would be required and perhaps at the same time a barrier would be established between the continued development of a program and its operational use. This appears to be the case for at least one proposal for a network-available version of our CONGEN program. The DENDRAL project has undertaken a very laborious conversion of CONGEN from its native LISP implementation to one in MAINSAIL to achieve a level of exportability for lack of other immediately available mechanisms. Other aspects of this approach involve security and privacy. Some of the data used with these programs are sensitive (patient records, or private, unpublished information on chemical structures, etc.). Having such a public access as over a network can create problems in protecting these data; and individual user groups may prefer to run the programs on machines which are under their local control. Finally, since many of these tools are in the research domain, it is not clear that they would be cost-effective in a commercial environment.

3.3.2.2 MACHINE-INDEPENDENT LANGUAGE IMPLEMENTATION

An ideal which has been long sought for program sharing is to develop languages with "universally" accepted standards and which are implemented in machine independent ways so that programs running on one machine environment will run in another with a minimum of conversion effort. This of course involves both language implementation and application program implementation concessions to achieve effective machine independence. We are working on a machine independent version of the SAIL language called MAINSAIL now to experiment with these sorts of issues. Our detailed plans for MAINSAIL development are given below including the possibility of special microprogrammed machines which may most economically and efficiently run MAINSAIL. Practically speaking, the machine-independent language approach is best-suited to the design of new program systems; and in the particular case of MAINSAIL, to those that can be effectively expressed by means of an ALGOL-like language. For existing programs, an extensive conversion would be required. We are still exploring the full range of implications of language choice for AI programs such as are being developed on SUMEX but it is likely that MAINSAIL cannot be a universal substitute for the full range of languages (including LISP) useful for these programs in both operational use and on-going development. MAINSAIL is nevertheless a definitive step toward understanding the requirements, advantages, and costs of machine independent systems. It may offer a useful base for implementing all or parts of new systems as well as for the ultimate reengineering of existing systems as they become fully operational.

3.3.2.3 EXPORTABLE (PDP-10) SYSTEM

An alternative view is that with the dramatic downward plunge of hardware costs, the costs of software development should play a larger and larger role in determining software/hardware optimizations. An attractive solution involves a PDP-10-like machine which could run the existing software intact and which could be made available for a reasonable cost to interested user (or network) groups. Since the machine could run the native operating system and language in which the program was developed, the initial conversion would be minimized and future developments (either conceptual or for improved efficiency) would be readily incorporated. Furthermore, a given user group could (perhaps with a change of microcode or system) run programs from various PDP-10 environments. By using network communication facilities, such satellite machines could retain contact with central development efforts, share files or data bases where appropriate, and provide a means for cost-effective incremental expansion by adding more such satellite machines or upgrading to a larger PDP-10 configuration when usage justifies. In this sense, this option is really a variant on the first network option using a more flexible hardware capability which can adapt better to individual program and development group/user community needs.

This approach may be best suited for this intermediate stage in AI program development where continued research and improvement is going on while extensive operational access is demanded. An economical export by this means defers the need for reprogramming until the design is fully stabilized and ready to be "cast in concrete". Nevertheless, even if the host machine is very inexpensive, in the long term if a factor of 10 improvement in speed or the number of users supported is possible by reprogramming, then a reimplementation will likely be warranted eventually as development tapers off and more and more users demand efficient production runs.

3.3.3 EXPORTABLE MACHINE PLANS

Because of the already large effort that has gone into other existing software systems we are attempting to export, the "exportable machine" option may offer a substantial advantage in minimizing conversion efforts, maintaining contact with program development groups, and offering a cost-effective way for even relatively small groups to use these programs. This is particularly important in just moving from the strictly developmental phase into a combined development/refinement/operational stage.

For our purposes, such a machine could be either a hardware-designed PDP-10 or a microprogrammed emulation of this machine. As a tentative functional configuration we would like the machine to perform at about the speed of a KI-10 with several users including:

- PDP-10 instruction set and "BB&N" paging facilities
- at least 256K logical address space
- 256K physical memory size (36 bit words, < 1 microsecond cycle)
- memory interface for swapping device and small file system including at least 200M bytes of disk storage
- facilties for about 16 terminals
- 200-300 lpm printer
- slow tapes
- some kind of external bus interface (I/O bus, UNIBUS, etc.)
- facilities for network communication connections

The cost for such a system (CPU, memory, and minimal peripherals) should ideally be in the range of $50,000 - $100,000. This may be below the initial announcement price for such machines but should represent realistic longer term pricing possibilities. A number of vendors may be working on the planning stages of such a machines which could be announced within the next 18 months. We budget for an initial version of such a machine at $200,000 based on very general pricing estimates (noting also that no vendor announcement has been made). The detailed alternatives and plans for this acquisition will be reviewed with the AIM management committees before implementation.

The detailed requirements for integrating such a machine into the SUMEX-AIM resource are also necessarily vague since this will depend on needed operating system and user support changes to accommodate the reduced size and perhaps different memory management system (paging). These changes may also reflect themselves in modifications for the language support underlaying the programs we want to export. We expect to track these developments closely during the first year of the follow-on grant and to formulate a plan for acquiring such a machine for experiments in packaging our AI programs for export. We will only be able to assess the required level of system software work when the details of the vendor systems become known. The budgetary details are discussed in the "justification" section of the five year budget plan.

These kinds of machines may also offer an effective way to incrementally expand the capacity of facilities like SUMEX and we will review them in this context as well (see the discussion of facility hardware upgrade plans on page 62). The main issues arising in coupling such satellite systems to the central facility as independent machines involve managing a distributed file system,

convenient terminal routing, and allocating users between machines. These are all manageable problems within existing technology such as we employed in developing the initial dual processor implementation. Since we are operating on fully amortized hardware, the indicated time table is driven by the real costs of system software modernization and compatibility of maintenance. Local users will be less injured by persevering with dated systems than a wider community to which software must be efficaciously exported in a contemporaneous idiom.

3.3.4 MAINSAIL DEVELOPMENT PLANS

The on-going MAINSAIL development effort was described earlier as part of our detailed progress report. A summary of language features can be found in Appendix III on page 231 (see Book II). This section summarizes the planned directions for future MAINSAIL developments. These efforts have two complementary thrusts: 1) development as a programming system and research tool and 2) demonstration of implementations for additional target systems. The first area is independent of what machines are used as hosts and seeks to explore the design ramifications, programming techniques, and advantages and costs of machine independence. The second area addresses the acquisition of practical experience in the export and use of MAINSAIL on real systems and the issues involved in gaining user acceptance of MAINSAIL as a programming tool.

3.3.4.1 DEVELOPMENT MANAGEMENT

In the early phases, the design for MAINSAIL was developed by Mr. Wilcox with a range of community inputs collected in relatively informal exchanges. These have included discussions with the designers of the SAIL language, studies of other languages (PASCAL, ALGOL-60/68, and SIMULA in particular), comments on our preliminary design documents from interested groups, presentations and discussions at several DECUS symposia, and community experimentation and critique of evolving MAINSAIL implementations. Our network connections have been invaluable in this regard, providing access to our documents, allowing rapid responses to suggestions, and providing a means for network collaborators to experiment with MAINSAIL on their own machines as implementations have become available. As MAINSAIL achieves a more operational status and we receive feedback from a larger community, we will reexamine many of these initial design decisions based on criteria of generality and effective portability as well as community acceptability. In this process we will formalize our user community contacts to take better advantage of their suggestions for system evolution and for effective system maintenance. We will, of course, provide a mechanism for reporting community comments (most easily done via networks) and may organize workshops or participate in other meetings to disseminate and discuss MAINSAIL. The AIM Executive Committee will play a key role in advising about development plans and making priority trade-offs within our limited available resources.

3.3.4.2 LANGUAGE DEVELOPMENT

Interrupts: We are currently investigating the implementation of both deferred and immediate interrupt facilities for MAINSAIL to give the ability to stop a program in the midst of execution, communicate with an interrupt-driven i/o device, or synchronize cooperating processes. A key issue is how to coordinate interrupt control transfers with on-going dynamic memory and storage management. This is particularly critical for immediate interrupts as may be

needed for real time applications. It may be necessary to restrict the range of language facilities available during such interrupts. We will continue these studies and implement appropriate interrupt handling support.

Concurrency: The current implementation of MAINSAIL has been designed with concurrency in mind, and appears to provide a solid base. We must complete the definition of the role of concurrency in MAINSAIL, then specify a set of primitives needed to support concurrency. There will then be an efficient implementation of these primitives including a convenient and flexible user interface.

Minimize runtime checking: Much of the code produced for runtime checking could be eliminated if the compiler "understood" more about the program. We propose to give MAINSAIL the ability to verify that certain conditions are met within the program so that more checking can be done at compiletime, and less at runtime. This involves exploration of what features MAINSAIL should include to allow the programmer to help in this process.

LEAP: LEAP is a facility in SAIL which provides an associative data store to allow the retrieval of data based on the partial specifications. We have encountered a number of prospective MAINSAIL users who have used and feel a need for LEAP. We plan to investigate the most useful features in LEAP which should be incorporated into MAINSAIL. It should be pointed out that many of the facilities of LEAP can easily and efficiently be coded in MAINSAIL using RECORD's.

3.3.4.3 COMPILER DEVELOPMENT

Increase speed of compilation: There is much room for improvement in the speed of compilation. The current version was designed for flexibility rather than efficiency. Most important is a close look at the symbol-table lookup, for that is where (the first pass of) the compiler spends most of its time.

Improve error detection and recovery: The compiler's error detection and recovery is now rather primitive. In general the entire edit-compile-debug loop should be streamlined for user convenience. We propose the utilization of a text editor as an integral part of compilation, so that MAINSAIL can automatically switch between compiling and user editing.

Machine-Independent code optimization: The first pass of the compiler produces an intermediate language which is the same for all target machines. This intermediate language is simply a recoding of the source file into an assembly-like language which reflects the properties of MAINSAIL. Various machine-independent transformations could be carried out on this intermediate text to translate it into an equivalent but more efficient representation of the source program.

Machine-dependent code optimization: The MAINSAIL code generators, themselves being MAINSAIL procedures, can be more readily written to utilize

complicated algorithms and data structures if necessary to generate efficient code. At present, the primary hurdle to a thorough analysis of the intermediate code by the code generators is the lack of a "look ahead" facility. We propose adding to the second pass the ability to build a machine-independent structure, on the procedure level, which can be interrogated by the code generators prior to generating code for a procedure. This would allow the code generators to make decisions based on a global knowledge of a procedure.

3.3.4.4 RUNTIME DEVELOPMENT

The runtime system is composed of modules which support the code generated for a user module. A single small module, called the kernel, is permanently resident, while all other modules are swapped as necessary. The modularity of the runtime system is what allows MAINSAIL to run in a small address space.

Optimize system modules: To a large extent, the efficiency of the system modules determines the efficiency of user programs. Thus it is well worth our time to optimize these modules. We propose to develop some modules which measure system performance. These would also be made available to users to help them evaluate their programs. A profile of a program, reporting how many times each statement is executed, is also proposed.

The primary use of these performance measurements will be for the tuning of memory allocation, swapping and garbage collection. MAINSAIL is largely independent of the exact strategies utilized, thus providing much leeway in working with alternate approaches. These algorithms need to be separately tuned for each implementation.

Virtual data space: MAINSAIL now supports the swapping of control sections, which could be considered a form of virtual control space. We are interested in studying whether this same form of support can be extended to data. Now that MAINSAIL can support a virtually unlimited control space (by breaking the program into modules), an implementation will be limited primarily by the amount of data which must be resident. We propose to add facilities to the language which allow the user to help structure the data so that it can be efficiently moved between memory hierarchies.

Support data operations: Machines which do not directly support the data types which MAINSAIL offers will need additional support modules. In particular, we need to write machine-independent modules to perform arithmetic on long integers, reals and long reals.

Runtime certifier: We will need a runtime certifier, i.e., a set of modules which give new MAINSAIL implementations a thorough workout, comparing the results with those obtained from running MAINSAIL on other machines. We have been using the compiler for this purpose, but it does not exercise all facilities of MAINSAIL, e.g., real and long real.

3.3.4.5 DEBUGGING SYSTEM DEVELOPMENT

We feel an effective and integrated debugging system will play a key role in the utility of MAINSAIL. Our goal is to provide interactive debugging capabilities comparable to those of INTERLISP which can significantly increase programming productivity. The combination of comprehensive debugging facilities with efficient production execution will help bridge the gap between program development and operational use.

The basic approach involves the integration of the now distinct phases of source text editing, compilation and execution. An internal representation of the program will be maintained which can serve a variety of purposes. This representation will be interpreted during debugging so that MAINSAIL can monitor execution and interact with the user in a manner which reflects the program structure. Errors can be corrected by editing this structure, and execution continued with no need for recompilation. Program text can be generated from the structure in a standard format, including the original variable names.

Machine code can be generated from this same structure, and compiled and interpreted code intermixed during execution. This provides fast execution of debugged modules along with interpreted execution of modules under scrutiny. Interpreted execution will allow for the interrogation of variables, setting and removal of break points, procedure trace, and single stepping. We plan to integrate these capabilities with a display terminal under the control of an editor, though the debugger will also operate from a hard-copy terminal. A split-screen facility will allow the program text to be viewed during execution along with any output from the program.

There are a number of difficult problems to be resolved concerning the relationship between the original source text (if any) and its internal representation which may be edited during debugging. Unlike LISP, the MAINSAIL syntax requires a significant amount of compilation before it can be put into a form which can be interpreted with reasonable efficiency.

3.3.4.6 DOCUMENTATION PLANS

Language manual: The currently available documentation for MAINSAIL consists of a preliminary language reference manual. It will be rewritten and expanded to be useful to users unfamiliar with SAIL.

Runtime manual: We will also provide a runtime manual which explains what happens during program execution. This information can be enlightening when designing a program, though its primary purpose is to document the machine-independent runtime system. This manual will also be necessary for the implementation of MAINSAIL on a new machine.

Code generation manual: A third manual, the code generation manual, will describe how to write code generators. This involves a description of the intermediate code, and how it is presented to the code generators. The goal is to

describe the code generation process in sufficient detail to allow any user to write a complete set of code generators. In this way the burden of implementing MAINSAIL on new machines can be dispersed.

System implementation manual: The system implementation manual will describe how to write the machine-dependent parts of the runtime system. This manual will describe what procedures need be written, and the data structures and other procedures with which they interact. It will also describe all the parts of MAINSAIL, how they fit together, and how to build a new system.

3.3.4.7 MAINTENANCE AND DISTRIBUTION PLANS

The maintenance and distribution of MAINSAIL could easily overwhelm us if we do not carefully plan for it. This is a good opportunity to bring someone else into the project, since it presents the chance to become familiar with the inner workings of the system.

Local experts: Each site must have a local expert who can repair errors in the machine-dependent portions and make patches to the machine-independent parts prior to receiving a new version which incorporates the changes. Another role for the expert would be that of liaison between the local user community and SUMEX. Questions and bug reports should first be directed to the local contact, and then directed to SUMEX in a form standardized across all sites.

SUMEX liaison: As MAINSAIL begins to be used at a number of sites, we would expect the number of inquires from potential users to rise to the point where it could require an inordinate amount of time from the developers. We propose that an additional person be hired at SUMEX as a liaison for MAINSAIL. This individual must be capable of fixing bugs and generally keeping current versions of the system healthy. The liaison will keep in touch with the local experts, and pass to them any necessary updates. This involves making tapes and sending them through the mail; editing the documentation, overseeing its printing and distribution; responding to inquiries from potential users; consulting with new users concerning program design (but not actually writing user's programs); and new user orientation.

3.3.4.8 PLANS FOR ADDITIONAL IMPLEMENTATIONS

The current implementations are for the PDP-10 and PDP-11. These give us experience on medium and small scale machines. We plan to hold off on introducing additional implementations until we have received sufficient feedback from these. It appears that the orchestration of parallel implementations on a wide variety of machines will rival the technical problems.

We have surveyed a large number of computer systems while designing MAINSAIL. Most of these are known to us only through manuals, so that further study will be necessary to determine how well a particular system could support MAINSAIL. Among the machines surveyed are: IBM (360/370, Series/1), CDC (6000 Series, 7600), UNIVAC (1100 Series), Texas Instruments (990), Honeywell (Level 6), Varian (V70), Hewlett-Packard (3000 and 2100), Data General (NOVA, ECLIPSE), Interdata (16 and 32 bit series), SEL (32), Harris (Slash series), Burroughs (B1700) and MODCOMP. We plan to keep abreast of new computer announcements, since we are in the position of relatively easily providing software for emerging hardware.

Choices for target systems will be based on user demand and priorities established in consultation with the AIM management committees. We are projecting approximately two man-months to create a new implementation, though this will vary according to how well the target machine and operating system fit MAINSAIL, and the availability of a target system during the early design iterations. Additional time will be required to actually install the implementation at the target site, have it thoroughly tested, distribute documentation and make it generally available. There are, of course, problems in developing MAINSAIL for a machine to which we have no access. The code generators and operating-system interface can be written independently of the target machine, but the debugging of these will require access for a period of at least a few weeks. It would not be acceptable to implement a machine by sending tapes through the mail. There appear to be four possibilities: access over a network; access to a nearby machine for which MAINSAIL has been implemented; rent or borrow a machine for the duration of the development; emulation of the target machine.

3.3.4.9 MAINSAIL OPERATING SYSTEM PLANS

In the course of designing the operating system interfaces it has become apparent that MAINSAIL needs very little support from any machine-dependent operating system, at least with regard to the execution of a single program. We feel that in many cases we could provide our own stand-alone version of MAINSAIL for single-job environments. Technology seems to be pointing in the direction of less expensive computers which can be dedicated to a single user at a time, and these would be the initial target of our operating system.

In the context of a single-job system, MAINSAIL's primary need is a file system and device drivers. Once our primitive operating system is written in MAINSAIL, it should not be difficult to add monitor commands and utilities such as file manipulation. Of course the MAINSAIL operating system would be special purpose in that it would support a single language, with everything designed around that language. The main elements of our operating system would be the compiler, a text editor, the MAINSAIL runtime system, and the additional modules to support the file system and i/o.

MAINSAIL does not need a linker, overlay system, or loader (the swapping of modules takes care of those). Additional components of the system could simply

be added as new modules. A goal would be to design an open-ended operating system kernel which could be extended by the user as desired.

3.3.4.10 MICROCODED MAINSAIL MACHINE PLANS

We have thus far been discussing the achievement of portability by making MAINSAIL fit existing machines. If the reason for portability is understood as the desire to provide an economically viable way of distributing software, then another approach is to make the hardware fit MAINSAIL, and distribute the hardware along with the software.

We propose to design an "optimal" representation of MAINSAIL code for emulation by a microprogrammable computer; to purchase a suitable computer for MAINSAIL emulation; to implement MAINSAIL and the supporting microcode on this computer; and to evaluate the resulting system to determine the economic and technical feasibility of distributing such an integrated hardware-software programming environment. Details of our plans are given in Appendix IV on page 235 (see Book II).

We expect considerable improvement over implementations for existing machines which have been accommodated to less than optimal, and in some cases quite poor, instruction sets. Many benefits accrue from such an approach, and it is likely that microcoded hardware, specialized to a particular language or application, will play an increasingly important role in the development and operational use of future software systems. We expect a microcoded MAINSAIL to outperform other MAINSAIL implementations in much the same way that DELtran (a "directly executable language" (DEL) implementation for FORTRAN II) outperforms FORTRAN II(4). Initial measurements show that the DELtran representation is less than one fifth the size of the code generated by the FORTRAN-H optimizing compiler, and executes about five times faster.

MAINSAIL is perhaps better suited to the emulation approach than FORTRAN because of the locality of reference provided by procedures, records and modules. A preliminary DEL has already been designed for MAINSAIL, but further work is necessary before we can predict (or demonstrate) size and execution comparisons with standard implementations.

This work will complement the on-going implementations of MAINSAIL on conventional hardware. Thus we will be in a unique position to compare the two approaches.

The combination of a microprogrammed machine with the MAINSAIL operating system could result in a system optimized for the execution of MAINSAIL programs. As hardware costs continue to fall we see this approach as a realistic way of providing a powerful system at a low price. We are interested in determining

---

(4) See Hoevel, L. W. and Flynn, M. J., "The Structure of Directly Executed Languages: A New Theory of Interpretive System Support," Stanford Digital Systems Laboratory, Technical Note No. 108, Stanford University, March 1977.

whether a "soft" machine of this sort can be provided cheaply enough to serve as a basis for the export of software which presently requires extensive hardware facilities.

3.3.4.11 DEVELOPMENT OF PORTABLE SOFTWARE

We would like to see a collection of portable programs developed in MAINSAIL both to serve as examples of portable software, and to provide support to those sites which begin to rely on MAINSAIL as the primary programming resource. Such software development will also help us debug MAINSAIL, familiarize the programmers with it, and spread its use. We are aiming for the complete support of a stand-alone MAINSAIL implementation which is aligned with developing hardware trends, i.e. video displays and compact, relatively inexpensive computers and peripherals.

We do not now have the facilities to implement all of this software at SUMEX, and thus expect to collaborate with others in its design and implementation. It is imperative that the software be portable except possibly for certain well-defined modules which need support outside MAINSAIL (e.g., special device support).

Display editor: A MAINSAIL text editor is at the core of a number of planned developments. Our interest is centered around a display-oriented editor because of its clear superiority over hard-copy editors. The TV-EDIT program now in use at Stanford and a few other sites is an excellent base of development, especially since it is written in SAIL. We would like to see additional features added to what TV-EDIT now possesses. Our intended applications for compilation and debugging require a split-screen facility, and a multi-file capability. It must direct all communication with the display through a display package, as described below. This separates the editing functions from the display functions, so that the editor is independent of the display and hence can be used with a variety of displays.

Display package: A display package is necessary as part of the editor, and is also important as a package for use by other programs. The display package will accept standard commands to control a display terminal. It must be smart enough to simultaneously maintain several areas on the screen. Such a package will be machine-independent (as much as possible), but have terminal-dependent modules which feed the terminal hardware commands to effect the machine-independent commands. It should be able to drive a hard-copy terminal as if it were a limited display terminal.

Graphics package: Similar to the display package is a graphics package for drawing pictures on a graphics display device. This package would allow for the description of pictures, the choice of display device, and the display of the pictures. This package would be machine-independent and display-independent. The OMNIGRAPH system developed by Sproull at NIH may form the basis for this package.

Document preparation: A simple document preparation program would serve as

the "back end" to the display editor. We feel that much of the work of current document programs could be provided by the editor in a form providing instant feedback. Thus the primary purpose of the document program would be to provide global processing, e.g., to generate a table of contents or index, and fill in symbolic references with appropriate chapter or section numbers.

Math and statistics packages: MAINSAIL currently has a mathematics package with trigonometric and logarithmic functions. These functions need additional testing for accuracy, and should be augmented with other functions, e.g., a random-number generator. There is also a need for a statistics package.

4 AVAILABLE FACILITIES

The existing SUMEX-AIM computer and communications configurations have been described in earlier sections. The number of personnel to support this follow-on work will remain at approximately the same level as before so no additional office space will be required. We anticipate no changes will be needed for the machine-room facilities.